Hints/Tips

1. Remember motivation comes from within. Take responsibility for your own learning.
2. Take notes from each presentation and activity.
3. If you don't understand something, ask questions.
4. Participate in course discussions.
5. Practice what you've learned to keep your skills sharp.

Activity 2

Activity Overview

This activity is all about simple operations using arrays. While this activity involves two C# classes, the actual requirements that need to be fulfilled are very simple, but will require some creativity to finish. You will also learn to use runtime exceptions as part of a very standard practice in C# programming: validation of method arguments.

Activity Hints and Tips

Just like the first activity, focus your efforts. Unlike the first activity, however, try to focus on each class as a whole entity. Imagine how every time 'pushing' an object or 'popping' an object will affect the stack.
You may want to research implementing stacks as arrays on the web. There are a lot of previous implementations there that can give you a good idea of what to do.

Activity Instructions

1) Create new solution and C# Class Library project named as **Activity2** and delete ***Class1.cs*** file from it.

2) First add the existing ***ArrayBase.cs*** and ***ArrayStack.cs*** files (from **Week 1 Activity Code Files** folder) to your project.

3) If you're already familiar with inheritance, you'll note that generic class ***ArrayStack<T>*** is a child class of the generic parent class ***ArrayBase<T>***. Even if you're not, just focus on the methods listed below and you should still be able to fulfill the requirements. Some of the child class methods are dependent on the parent.

Note: Refer to the TODO markers within the source code for detailed instructions.

4) Work on the activity requirements concerning ***ArrayBase*** first. While the activity class will only test ***ArrayStack***, any behavior not properly implemented by the parent class will affect the child class.

5) The following behaviors need to be completed:

- **a)** ***public ArrayBase(int)*** – a *constructor* method that initializes the array structure with the specified size or a default size
- **b)** ***public virtual int IndexOf(T)*** – a method which finds and returns the index of the object which is similar to the specified argument object (by using ***Equals()*** method of the **System.Object** class)

  Note: The ***virtual*** concept will be explained later in this course.
- **c)** ***public virtual int Add(T)*** – a method which adds new object to the array and returns NOT_IN_STRUCTURE if array is full, or added element index if addition succeeded.
- **d)** ***public virtual void RemoveAt(int index)*** – a method which removes object from the array at given position or throws ***InvalidOperationException*** if removal is not possible. Remember to 'compact' the elements if **index** does not point to the last object.

6) The class ***ArrayStack.cs*** is a representation of a *stack* data structure. A stack is an LIFO (Last-In-First-Out) structure, meaning that the last object entering the stack will be the first item to be removed (It can also be viewed as a FILO – First-In-Last-Out).

7) In this activity the array ***storeArray*** from the class ***ArrayBase*** will represent the stack, with the 'bottom' being index 0. The variable ***currentCount*** counts the current number of objects present in the stack.

8) The following are methods you must complete; they represent common operations involving stacks:

- **a)** ***public virtual void Push(T)*** – Pushes an object onto the 'top' of the stack, assuming that the stack is not full and the object is not already in the stack. Throw ***InvalidOperationException*** if the stack is full.
- **b)** ***public virtual T Pop()*** – Removes an object from the 'top' of the stack and returns it. Throw ***InvalidOperationException*** if the stack is empty.
- **c)** ***public virtual T Peek()*** – Returns but does not removes the object at the 'top' of the stack or throws ***InvalidOperationException*** if the stack is empty.

Note: Access the **storeArray** elements from the derived class ***ArrayStack***, use ***ArrayBase*** class indexer property, e.g., ***base[index]***.

9) To test your solution for correctness, add new test project to your existing **Activity2** solution: File > New > Project…. and select Visual C# > Test > Unit Test Project. Name it as **Activity2.Tests** and make sure you are adding it to the existing **Activity2** solution.

10) In the **Activity2.Tests** project add test file ***Activity2_Tests.cs*** (from Week 1 Activity Code Files\ folder) and ***PlayerProfile.cs*** (it might be the finished version from Activity 1).

11) In the **Activity2.Tests** project add a reference to the project **Activity2**.

12) In order to activate unit tests open TEST > Windows > Test Explorer and click Run All – if all tests pass green, then your solution is correct. If some tests fail, try to infer why your solution behaves differently.

13) Inform the facilitator upon completion by committing solution **Activity2** to the TFS with comment “Finished Activity 2”.